Principles of Parallel Algorithm Design

(2)

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Topics

• Introduction to Parallel Algorithms
  – Tasks and Decomposition
  – Processes and Mapping
  – Processes vs. Processors

• Decomposition Techniques
  – Recursive Decomposition
  – Data Decomposition
  – Exploratory Decomposition
  – Hybrid Decomposition

• Characteristics of Tasks and Interactions
  – Task Generation, Granularity, and Context
  – Characteristics of Task Interactions.
Topics – con’t

• Mapping Techniques for Load Balancing
  – Static and Dynamic Mapping
• Methods for Minimizing Interaction Overheads
  – Maximizing Data Locality
  – Minimizing Contention and Hot-Spots
  – Overlapping Communication and Computations
  – Replication vs. Communication
  – Group Communications vs. Point-to-Point Communication
• Parallel Algorithm Design Models
  – Data-Parallel, Work-Pool, Task Graph, Master-Slave, Pipeline, and Hybrid Models
Characteristics of Tasks

• Tasks
  – Pieces of work decomposed

• Key characteristics
  – Task generation
  – Task sizes
  – Size of data associated with tasks

• Critically impact choice and performance of parallel algorithms
Task Generation

• **Static task generation**
  – Tasks can be identified a-priori
  – Typically decompose using data or recursive decomposition techniques
    • Matrix operations
    • Graph algorithms
    • Image processing applications
    • Other regularly structured problems

• **Dynamic task generation**
  – Tasks are generated as computation performed
  – Typically decompose using exploratory or speculative decompositions
    • Game playing
    • 15 puzzle
Task Sizes

• Uniform
  – Example: matrix-vector multiplication

• Non-uniform
  – Task sizes can be determined a-priori
  – Or not
  – Examples
    • In quicksort, size of each partition depends on pivot selected
    • In 15-puzzle, sizes of tasks are unknown
Size of Data Associated with Tasks

• Data associated with a task may be small or large compared to computation
  – Example
    • 15 puzzle: size(input) < size(computation)
    • Min: size(input) = size(computation) > size(output)
    • Qsort: size(input) = size(output) < size(computation)

• Implications
  – Small data: task can migrate to other processes easily and dynamically
  – Large data: ties the task to a process
    • Avoids excessive communication of task contexts
Characteristics of Task Interactions

• Different dimensions of task interactions
  – Static vs. dynamic
  – Regular vs. irregular
  – Read-only vs. read-write
  – One-sided vs. two-sided
Characteristics of Task Interactions

• **Static interactions**
  – The tasks and their interactions are known a-priori
  – Simpler to code
  – Example
    • Matrix multiplication

• **Dynamic interactions**
  – Timing or interacting tasks cannot be determined a-priori
  – Harder to code
    • Especially, using message passing APIs.
  – Example
    • 15 puzzle – a task with exhausted search space need to take another unexplored search space
Characteristics of Task Interactions

• Regular interactions
  – Having a definite pattern (in the graph sense) to the interactions
  – Regular interactions can be exploited for efficient implementation
    • Schedule tasks to avoid conflicts on network links

• Irregular interactions
  – Lack well-defined topologies
  – Modeled by graphs
Static Regular Interaction Pattern

• Example: image dithering
  – Pixel values on edges should be passed to adjacent tasks
    • Location of data to be sent to other tasks is deterministic regardless of input
Static Irregular Interaction Pattern

- Example: sparse matrix-vector multiplication
  - Interaction pattern varies depending on the input matrix
Characteristics of Task Interactions

• Read-only interactions
  – Tasks just read data associated with other tasks

• Read-write interactions
  – Tasks read, as well as modify data associated with other tasks
  – Harder to code
    • Requiring additional synchronization primitives
Characteristics of Task Interactions

• **One-way interaction**
  – Initiated and accomplished by one of the two interacting tasks
    • Read or write
    • Get or put
  – Somewhat harder to code in message passing APIs

• **Two-way interaction**
  – Requiring participation from both tasks involved in an interaction
    • Send and recv in message passing APIs
Mapping Techniques

• Decomposed concurrent tasks should be mapped to processes
  – Then, those can be executed on a parallel platform

• Mapping overheads
  – Communication
  – Idling (or serialization)

• Minimizing these overheads often represents contradicting objectives.
  – Assigning all work to one process
    • No communication but significant idling
  – Minimizing serialization – balance load among processes
    • Introducing communications
Mapping Techniques for Minimum Idling

- Mapping must simultaneously minimize idling and load balance
- Merely balancing load does not minimize idling
  - Two cases are balanced in loads, but (b) shows much idling
Mapping Techniques for Minimum Idling

• Static Mapping
  – Tasks are mapped to processes a-priori
  – Requirements
    • A good estimate of the size of each task
    • Even in these cases, optimal mapping may be NP complete
      – E.g., multiple knapsack problem

• Dynamic Mapping
  – Tasks are mapped to processes at runtime
    • Tasks are generated at runtime
    • Their sizes are not known

• Other factors influencing choice of mapping
  – Size of data associated with a task
  – Nature of underlying domain (e.g., NUMA)
Schemes for Static Mapping

• Mappings based on data partitioning
  – Block distribution
  – (Block-)cyclic distribution
  – Randomized block distribution
  – Graph partitioning
• Mappings based on task partitioning
• Hybrid (or hierarchical) mappings
Mappings Based on Data Partitioning

- Data partitioning + owner-computes rule
- Example: 1-D block distribution for dense matrices

```
row-wise distribution

<table>
<thead>
<tr>
<th>P_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
</tr>
<tr>
<td>P_2</td>
</tr>
<tr>
<td>P_3</td>
</tr>
<tr>
<td>P_4</td>
</tr>
<tr>
<td>P_5</td>
</tr>
<tr>
<td>P_6</td>
</tr>
<tr>
<td>P_7</td>
</tr>
</tbody>
</table>

column-wise distribution

<table>
<thead>
<tr>
<th>P_0</th>
<th>P_1</th>
<th>P_2</th>
<th>P_3</th>
<th>P_4</th>
<th>P_5</th>
<th>P_6</th>
<th>P_7</th>
</tr>
</thead>
</table>
```
Block Array Distribution Schemes

- Multi-dimensional block distributions
  - More tasks and data partitions \( \Rightarrow \) higher degree of concurrency can be possible

\[
\begin{array}{cccc}
P_0 & P_1 & P_2 & P_3 \\
P_4 & P_5 & P_6 & P_7 \\
P_8 & P_9 & P_{10} & P_{11} \\
P_{12} & P_{13} & P_{14} & P_{15}
\end{array}
\]
Block Array Distribution Schemes: Examples

• Multiplying two dense matrices $C = A \times B$
  – Partition the output matrix $C$ using a block decomposition.
  – Give each task the same number of elements of $C$
    • Each element of $C$ corresponds to a single dot product
    • Load balanced

• 1-D vs 2-D decomposition
  – Determined to minimize associated overhead
    • Communication, memory
Data Sharing in Dense Matrix Multiplication

\[
\begin{align*}
\begin{array}{c}
A \\
X
\end{array} & \begin{array}{c}
B \\
\end{array} & = & \begin{array}{c}
C
\end{array} \\
\end{align*}
\]

(a)

Required memory

\[
\frac{2n^2}{p} + n^2
\]

\[
\frac{n^2}{p} + \frac{2n^2}{\sqrt{p}}
\]
Cyclic and Block Cyclic Distributions

• If the amount of computation associated with data items varies
  – A block decomposition may lead to significant load imbalances

• A simple example of this is in LU decomposition (or Gaussian Elimination) of dense matrices
LU Factorization of a Dense Matrix

- A decomposition of LU factorization into 14 tasks

\[
\begin{pmatrix}
A_{1,1} & A_{1,2} & A_{1,3} \\
A_{2,1} & A_{2,2} & A_{2,3} \\
A_{3,1} & A_{3,2} & A_{3,3}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
L_{1,1} & 0 & 0 \\
L_{2,1} & L_{2,2} & 0 \\
L_{3,1} & L_{3,2} & L_{3,3}
\end{pmatrix}
\cdot
\begin{pmatrix}
U_{1,1} & U_{1,2} & U_{1,3} \\
0 & U_{2,2} & U_{2,3} \\
0 & 0 & U_{3,3}
\end{pmatrix}
\]

1: \( A_{1,1} \rightarrow L_{1,1}U_{1,1} \)  
6: \( A_{2,2} = A_{2,2} - L_{2,1}U_{1,2} \)
11: \( L_{3,2} = A_{3,2}U_{2,2}^{-1} \)

2: \( L_{2,1} = A_{2,1}U_{1,1}^{-1} \)  
7: \( A_{3,2} = A_{3,2} - L_{3,1}U_{1,2} \)
12: \( U_{2,3} = L_{2,2}^{-1}A_{2,3} \)

3: \( L_{3,1} = A_{3,1}U_{1,1}^{-1} \)  
8: \( A_{2,3} = A_{2,3} - L_{2,1}U_{1,3} \)
13: \( A_{3,3} = A_{3,3} - L_{3,2}U_{2,3} \)

4: \( U_{1,2} = L_{1,1}^{-1}A_{1,2} \)  
9: \( A_{3,3} = A_{3,3} - L_{3,1}U_{1,3} \)
14: \( A_{3,3} \rightarrow L_{3,3}U_{3,3} \)

5: \( U_{1,3} = L_{1,1}^{-1}A_{1,3} \)  
10: \( A_{2,2} \rightarrow L_{2,2}U_{2,2} \)
LU Factorization of a Dense Matrix

• Block distribution causes load imbalance (or idling)

\[
\begin{array}{ccc}
P_0 & P_3 & P_6 \\
T_1 & T_4 & T_5 \\
P_1 & P_4 & P_7 \\
T_2 & T_6 & T_8 & T_{10} & T_{12} \\
P_2 & P_5 & P_8 \\
T_3 & T_7 & T_9 & T_{13} & T_{14}
\end{array}
\]
Block-Cyclic Distributions

• Variation of the block distribution
  – Alleviate the load-imbalance and idling problems.

• Steps
  – Partition an array into many more blocks than the number of available processes
  – Assign blocks to processes in a round-robin manner
    • Each process gets several non-adjacent blocks
Block-Cyclic Distributions

- A cyclic distribution is a special case in which block size is one
- A block distribution is a special case in which block size is n/p
  - n is the dimension of the matrix and p is the number of processes
Randomized Block Distribution

- Sometimes, block-cyclic distribution causes load imbalance
  - Example: sparse matrix-vector multiplication
  - Diagonal processes are overloaded
Randomized Block Distribution

• Solution
  – Randomize tasks

Sparse matrix

Randomized

2D block distribution

\[
\begin{array}{cccc}
P_0 & P_1 & P_2 & P_3 \\
P_4 & P_5 & P_6 & P_7 \\
P_8 & P_9 & P_{10} & P_{11} \\
P_{12} & P_{13} & P_{14} & P_{15} \\
\end{array}
\]
Mappings Based on Task Partitioning

• In case of sparse data structures, block distribution-based mapping is more complex
  – Ex) physical phenomena simulation
  – Interaction is data dependent and irregular
  – Data is represented using graph

• (Task interaction) graph of data structures is a useful indicator
  – Work is number of nodes
  – Communication is the degree of each node

• Partition the graph
  – Assign equal number of nodes to each process
    • Balance work
  – Minimize edge count of the graph partition
    • Minimize communication
Mappings Based on Task Partitioning

• Partitioning a given task-dependency graph
  – When task-dependency graph is static
  – When task sizes are known

• Optimal mapping for a general task-dependency graph
  – NP-complete problem.

• Excellent heuristics exist for structured graphs
Mapping a Sparse Graph

- Task partitioning and mapping based on 1D block distribution

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Process 0

- C0 = (4, 5, 6, 7, 8)

Process 1

- C1 = (0, 1, 2, 3, 8, 9, 10, 11)

Process 2

- C2 = (0, 4, 5, 6)

17 communications
Mapping a Sparse Graph

- Task partitioning and mapping based on the task interaction graph

13 communications
Hierarchical Mappings

• Sometimes a single mapping technique is inadequate
  – For example, the task mapping of the binary tree (quicksort) cannot use a large number of processors

• Hierarchical mappings
  – Use a task mapping at the top level
  – Data partitioning within each task
Schemes for Dynamic Mapping

• Dynamic mapping (or dynamic load balancing)
  – Load balancing is the primary motivation for dynamic mapping.

• Effective when task size is unknown
  – Possible load imbalance

• Necessary when task generation is dynamic

• Styles
  – Centralized vs. distributed

• Considerations
  – In shared-address-space architecture
    • Associated data movement is implicit
  – In message-passing architecture
    • Effective when task size is larger than associated data size
Centralized Dynamic Mapping

- Processes
  - Master(s)
    - Manage group of tasks and assign task(s) to slaves
  - Slaves
    - When a slave runs out of work, it requests the master for more work

- Challenge
  - Master may become bottleneck for large number of processes (slaves)

- Approach
  - Chunk scheduling
    - A process picks up a number of tasks (a chunk) at one time
    - Selecting large chunk sizes may lead to significant load imbalances as well
    - Gradually decrease chunk size as the computation progresses
Distributed Dynamic Mapping

- Tasks are distributed among processes
- Processes exchange tasks to balance load
  - Avoids bottleneck in centralized schemes
- Four critical questions
  - How are sending and receiving processes paired together?
  - Who initiates task transfer?
  - How many tasks are transferred?
  - When is a task transfer triggered?
- Answers are generally application-specific
Topics – con’t

• Mapping Techniques for Load Balancing
  – Static and Dynamic Mapping

• Methods for Minimizing Interaction Overheads

• Parallel Algorithm Design Models
Minimizing Interaction Overheads (1)

• Factors affecting interaction overhead
  – Volume of exchanged data during interaction
  – Frequency of interaction
  – Spatial and temporal patterns of interaction
Minimizing Interaction Overheads (1)

- Maximize data locality
  - Where possible, reuse intermediate data
  - Restructure computation to reuse data promptly
- Minimize volume of data exchange
  - Minimize the volume of data communicated by carefully partitioning task interaction graph
    - Ex. Local sums for global sum
- Minimize frequency of interactions
  - Try to merge multiple interactions to one, where possible
    - Ex. Local sums for global sum
- Minimize contention and hot-spots
  - Use decentralized techniques
  - Replicate data where necessary
Example: Minimizing Interaction Overheads (1)

- Minimize contention and hot-spots
  - Example: dense matrix multiplication
    - Matrices are split into 16 pieces (Ai,j, Bi,j, Ci,j)
    - 2D block distribution
      - P0, P1, P2 and P3 will contend on A0,* at the same time

\[
\begin{array}{cccc}
A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\
A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\
A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\
A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \\
\end{array} \quad \times \quad \begin{array}{cccc}
B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\
B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\
B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\
B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \\
\end{array} = \begin{array}{cccc}
P_{0} & P_{1} & P_{2} & P_{3} \\
P_{4} & P_{5} & P_{6} & P_{7} \\
P_{8} & P_{9} & P_{10} & P_{11} \\
P_{12} & P_{13} & P_{14} & P_{15} \\
\end{array}
\]

- Solution: rotate the access sequence of A0,* in each process
  - P0 begins on A0,0 while P1 begins on A0,1, …
Minimizing Interaction Overheads (2)

• Overlapping computations with interactions
  – Non-blocking communications
    • E.g., non-blocking send in MPI
  – Multithreading
  – Assisted by Prefetching hardware

• Replicating data or computations

• Using group communications instead of point-to-point primitives
  – Group communications are optimized to minimize interactions

• Overlap interactions with other interactions
Minimizing Interaction Overheads (3)

• Overlap interactions with other interactions
  – Example: broadcast interaction

(a) A naïve broadcast

(b) Efficient broadcast

(c) Overlapping four naïve broadcasts
Topics – con’t

• Mapping Techniques for Load Balancing
  – Static and Dynamic Mapping
• Methods for Minimizing Interaction Overheads
• Parallel Algorithm Design Models
Parallel Algorithm Models (I)

• A way of structuring a parallel algorithm
  – Decomposition
  – Mapping technique
  – Applying strategies to minimize interactions
Parallel Algorithm Models (2)

• Data parallel model
  – Each task performs similar operations on different data (data parallelism)
  – Tasks are statically (or semi-statically) mapped to processes
  – Data decomposition
    + task-process mapping based on data partitioning

• Task graph model
  – Use a task dependency graph and its interrelationships
    • To promote locality or to reduce interaction costs
    • To exploit task parallelism
  – All decomposition schemes + mapping based on task partitioning
Parallel Algorithm Models (3)

• Work pool model
  – Static/dynamic task generation → task pool
  – Dynamic mapping of tasks onto processes
  – Centralized/decentralized pool of tasks

• Master-slave model
  – One or more processes generate tasks → master(s)
  – Allocate it to worker processes
  – Mapping may be static or dynamic.
Parallel Algorithm Models (4)

• **Pipeline / producer-consumer model**
  – Pass a stream of data through a succession of processes
  – Tasks (computations) are static but data are dynamic
  – Each performs some task on it

• **Hybrid model**
  – Multiple models applied hierarchically
  – Or, multiple models applied sequentially to different phases
References

• “Introduction to Parallel Computing”, by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar, Addison Wesley, 2003

• “COMP422: Parallel Computing” by Prof. John Mellor-Crummey at Rice Univ.